

Blumengarten

Tassilo Tanneberger

September 8, 2019

1 Ideen der Algorithmen

Ist eine Sammlung an prinzipiellen Ideen die ich mir gemacht hatte bevor ich anfangen hatte zu Implementieren.

1.1 Bewertung von Farben

w_i ... Wichtung der Präferenz

w_f ... Finale Wichtung der Farbe

N ... Anzahl der Präferenzen mit dieser Farbe

$$w_f = \sum_{i=1}^N (w_i^2 + 1) \quad (1)$$

Diese Gleichung für die Wichtung hat den Vorteil dass durch das Quadrat die Wichtung des Nutzer mehr gewertet wird als wie oft er diese Farbe angab.

1.2 Priority Places

Jeder Platz auf dem Blumenbeet hat eine gewisse Anzahl an Nachbar Blumen n . Wobei n genau den Wert dieses Platzes repräsentiert. Dadurch dass die Anordnung auch bis zu einem gewissen Grad symmetrisch ist kann man nun anfangen die "wichtigen" Farben nun an wichtige Plätze packen. Somit ist mehr oder weniger eine feste Reihenfolge der Platzierungen gegeben wo zuerst die wichtigsten Farben (mit dem höchsten w_f) gesetzt werden.

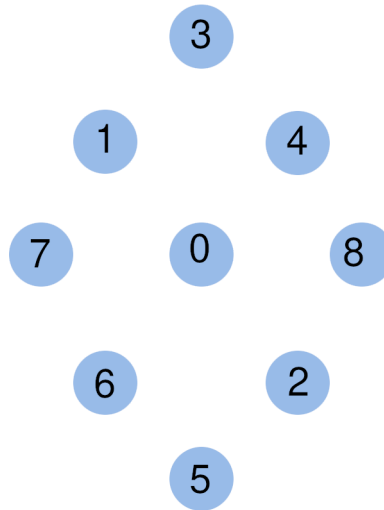


Figure 1: Reinforme in der die Plätze vergeben werden

Wobei hier zu beachten ist das 0, 1, 2 Immer mit den höchst gewichtesten Werten Platziert werden. Weil auch wenn alle 7 Farben genutzt werden sollen bleiben die $2 = 9 - 7$ für 1 und 2 übrig.

2 Implementierung

Alles wurde einfach aus einfachheits Gründen in Python3.7 Implementiert. Python als highlevel sprache ist einfach Ideal für solche klein Projekte weil man sich z.B nicht um Memory Leaks, nervende Seq. Faults und ähnliches kümmern muss.

2.1 Sorted by Priority

2.1.1 User Input

Ich habe den Nutzer input etwas anders Designed indem ich zuerst die Preferenzen angeben lasse und danach frage ob er noch andere Farben möchte das macht mehr Sinn aus den einfachen Grund weil man somit Konflikte vermeiden kann. Ein Beispiel der Nutzer sagt er möchte 2 Farben haben und sagt dann Rot-Blau-3 und Grün-Blau-2 und er hat einen Konflikt erzeugt weil er 3 Farben in den Präferenzen anbietet. anstelle nur von 2.

2.1.2 Bewertung der Farben

Implementierung der Bewertung.

In Datei ./Blumengarten/Field.py

```
def __calculate_value(self):  
  
    self.colour_values: List[int] = [0] * 7  
  
    # Element: 0 First Colour, 1 Second Colour, 2 Weight  
  
    for element in self.user_input.preferences:  
  
        self.colour_values[element[0]] += element[2] ** 2 + 1  
        self.colour_values[element[1]] += element[2] ** 2 + 1
```

Wobei preferences eine Liste mit Tupeln der Form (color1, color2, weight) ist.

2.1.3 Generieren der anderen Farben

Diese Methode erzeugt die zusätzlichen Farben zufällig. Zufällig einfach aus dem Grund um noch ein wenig Variatät ins Spiel zu bringen.

```
def generate_random_colors(self):  
    self.additional_random: List[int] = []  
    self.final_colors: List[int] = self.user_input.used_colors  
    if self.user_input.colors_count > 0:  
        for x in range(self.user_input.colors_count):  
            ran: int = random.choice(self.user_input.open_colors)  
            del self.user_input.open_colors[self.user_input.open_colors.index(ran)]  
            self.additional_random.append(ran)
```

2.1.4 Plazieren

Als erstes werden die oben angesprochenen freien besten Plätze statisch mit den besten Farben besetzt wie in der Reihenfolge in Figure 1 beschrieben siehe Funk-

tion placespare(). Wenn dieser Prozess abgeschlossen ist müssen wir alle Übrigen Farben noch unterbringen. Übrige Farben sind nun alle Zufälligen und nicht besten 2. Von diesen Farben wird es jeweils nur 1 geben. So nun wir wollen logischerweise hier auch noch Punkte Sammeln bedeutet wir müssen uns für die Restlichen auch noch schauen wo wir sie am Klügsten hinplatzieren. Dafür gibt es folgende Einfache Schrittfolge:

1. Indices aller umliegender Plätze heraus finden
2. Nach den Farben schauen die auf diesen Plätzen sich befinden
3. Dann wird die Farbe mit der Höchsten Synergie mit umliegenden Farben ermittelt
4. Farbe wird ins Feld geschrieben und aus den verfügbaren Farben heraus gestrichen

Synergie Funktion

```
def query_highest_syn(self, color: list, already_used: list, possible_colors:
    ratings: List[int] = [0] * 7

    for x in possible_colors:
        if x in self.user_input.used_colors and x not in already_used:
            for c in color:
                ratings[x] += Field.get(self.field.prepared, (c, x))

    if ratings == [0] * 7:
        i: int = random.choice(possible_colors)
    else:
        i: int = ratings.index(max(ratings))
    return i
```

Er Iteriert einfach durch die Verschiedenen Möglichkeiten machen weil es sehr übersichtlich viele Farben sind. Er Schaut ob die Farbe noch zur Verfügung steht also in possiblecolors gelistet ist und nicht usedcolors ist. Dann schaut er sich die umliegenden Farben (color) an und fragt die Wichtung zwischen diesen beiden Farben ab wenn es keine gibt ist der Wert einfach 0. Zum Schluss müssen wir dann noch schauen ob es überhaupt Synergien gab gab es keine Wird eine zufällige Farbe gewählt. Sonst der Wert der insgesamt die höchsten Synergien erreichte.